Understanding Q-Learning in 20 Questions

Andrew Gibbs-Bravo | Mike Cluley

Software Agents, MSc Data Science

School of Mathematics, Computer Science and Engineering

City, University of London

Abstract

This paper demonstrates how a Q-Learning agent can be trained to achieve 100% accuracy and performance roughly in line with a decision tree on a reduced scale 20 questions environment. We show that the reward function used is far more important for converging towards optimality than the hyperparameters although that they still impact performance. Finally, we discuss parallels with associative learning and apply a Deep-Q Network.

1 Introduction

Reinforcement learning has long been applied to path searching problems with simple state spaces.[1] A real-world problem for search engines and dialogue systems is that they must optimize two goals. That is, determining the point at which enough information has been gathered to stop searching and to make a prediction in order to achieve a high level of accuracy.

The classic 20 questions game is an abstraction of this concept. The game consists of two players: the Guesser (G) and the Respondent (R) where R selects an animal and G asks binary questions to guess the R's animal. There are 2 challenges: to accurately predict the animal and, do so in a minimum number of questions. 20 Questions represents a reasonable environment to evaluate deep reinforcement learning for dialogue systems[2] and knowledge acquisition with Microsoft Research applying deep RL to 20 Questions in two recent papers from 2018.[3] [4]

Example Game

- Respondent: Selects **swan** (unknown to Guesser)
- Guesser: Is it aquatic?
  Respondent: Yes
- Guesser: Is it Airborne?
  Respondent: Yes
- Guesser: Is it a Swan?
  Respondent: Yes, that's correct!

We are focusing on a variant of the classic game (detailed in the "Problem Definition" section) in order to evaluate the efficacy with which the agent can learn to ask questions based on its knowledge representation. The reinforcement learning algorithm we will apply is tabular Q-learning while also considering extensions to Deep Q-Learning. Given the questions are sequential and binary we can evaluate the agent's performance against a decision tree while noting that in the simple and complex environments the agent does not have access to the underlying knowledge base while the decision tree does.

1.1 Scope of the Problem

We have programmed a 20 Question game environment from scratch and will consider three levels of complexity:

1. Simple case: consists of four animals and two questions used to explain the problem, demonstrate how the Q-matrix updates, and how the reward matrix is represented. The state space consists of 27 discrete states composed of a combination of the status of the possible questions plus the game state. The action space consists of the remaining possible questions and animals to guess. (Q-matrix dimensions: 27 states x 6 actions)
2. Complex case: consists of 19 animals and 6 questions used in the gridsearch to demonstrate how the performance changes with changes in model hyperparameters. The state space and action space are the same as the simple case although consists of 2,187 discrete states therefore the Q-matrix dimensions are 2,187 states x 25 actions.
3. Continuous / Bayesian case: consists of a reformulation of the environment where the states are the probability a given animal is correct plus the question number in order to facilitate learning for the Deep-Q Agent.

2 Problem Definition

In order for our Q-learning agent to effectively learn how to navigate the environment, the problem needs to be precisely defined. The agent makes its action based on the signal contained in the state which should encapsulate "past sensations compactly, yet in such a way that all relevant information is retained".[5] States which accomplish this are said to follow a Markov Decision Process (MDP) meaning "The future is independent of the past given the present"[6] which is very important in effectively implementing Q-learning.

2.1 The Reward Function and the State Transition Function

The original motivation for this report is the well-known 20-Question Game. However, for the purposes of illustrating principles and Q-Learning set-up, a "small" 2-Question, 4-Animal game will be used. For quantitative and qualitative analysis, a "complex" 6-Question, 19-Animal game will be used.

2.1.1 Definition of the "Small" Game

The "small" game used for illustration in this report is a "2Q4A game" i.e. it comprises 2 questions with 4 possible animals combinations shown in the grid.

		Is animal airborne?	
		Yes	**No**
Is animal aquatic?	**Yes**	Swan	Dolphin
	No	Fruitbat	Cavy

2.1.2 *Reward Function*

The value of an agent moving to a new state by taking an action from its current state is defined by the reward function: $r_{t+1} = r(s_t, a_t)$. In the game the initial/baseline reward function values are detailed in the table and shown in following graphs and R-matrix. An alternative reward function was also run as part of the gridsearch.

Reward Function	r=Reward
Ask question	-1
Guess animal wrong	-10
Guess animal right	5

2.1.3 *Defining State Code IDs*

We constructed a unique identifier code for each state. This state code id is used in examples in this paper. In the small game the code is a concatenation of 4 fields – see table for components. A longer code is required for games with more questions. Example state codes:

0000 = Initial start state, no questions asked, no guesses made.

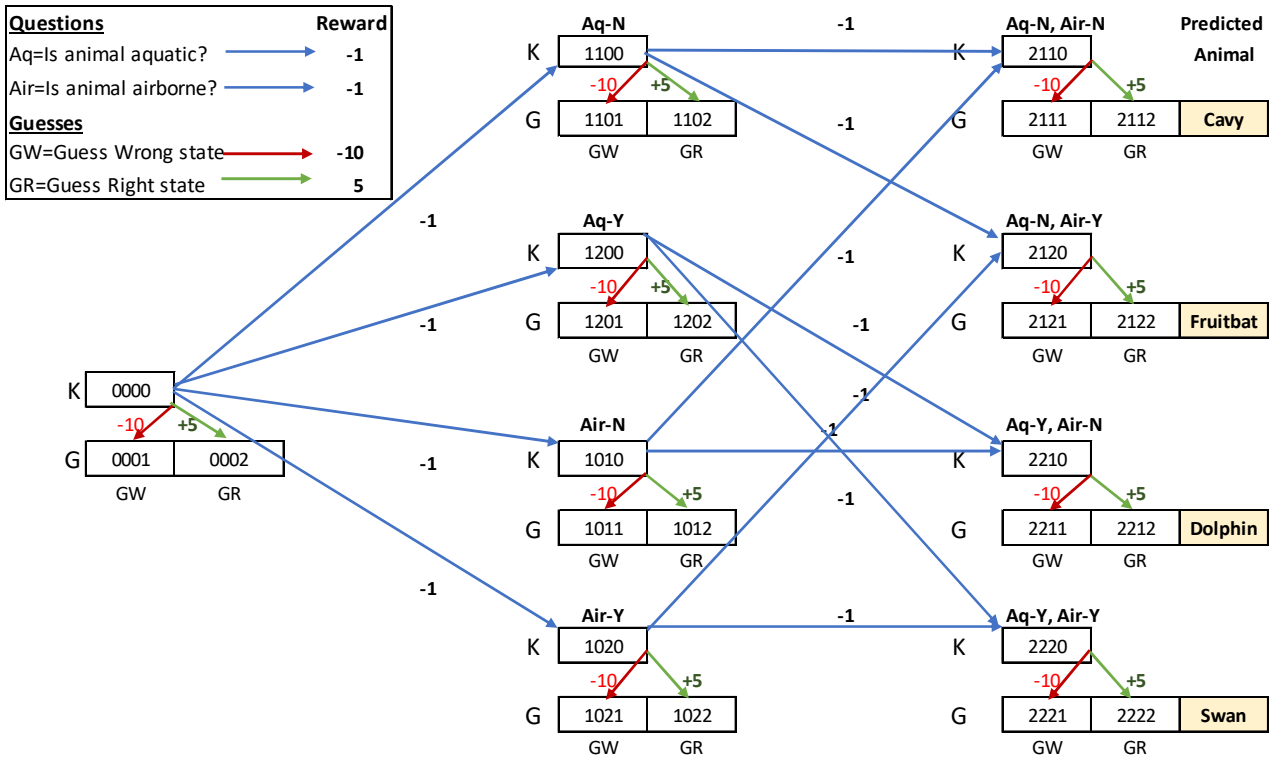
1020 = 1st Round. Question A not asked yet. Question B asked, answer "yes". No guess made yet.

NB: In the actual python code the corresponding values of 0,1,2 are -1, 0, 1.

Code ID Components	Value / Description
Round	0=Start state
	1=1 action taken, ask question or guess
	2=2nd action taken, ask question or guess
Question	0=Question not yet asked
	1=Question asked, answer "no"
	2= Question asked, answer "yes"
Guess	0=Guess not made yet
	1=Guess made, guess wrong
	2=Guess made, guess right

2.1.4 *Graphical Representation of Problem*

For illustration, at state 0000 the agent can travel to 1 of 6 states. The agent makes a guess leading it to one of two possible guess states (Guess Wrong=0001, reward=-5 or Guess Right=0002, reward=+10) or asks 1 of 2 questions leading to 1 of 4 states depending on each answer response, which all have a reward=-1 (e.g. ask if aquatic, answer Yes=1200, or ask Aquatic, answer No=1100, or, ask if airborne, answer Yes=1020, or ask airborne, answer No=1010). Guessing an animal results in a terminating state and the end of a game.



The graph sets out all 27 states for the small 2Q4A game and their inter-relationships

The arrow colour also indicates the different rewards associated with moving to different states

2.1.5 State Transition Function

The transition function is deterministic where a given action will always result in a transition to the same following state assuming the same correct animal. Because questions cannot be repeated the transition state space is relatively sparse.

The table on the right details the State Transition Function for the 27 states in the "2Q4A Small" game with commentary. The State Code ID is also deconstructed to show features of each state detailed.

We have defined 2 types of state:

- Knowledge State (K): indicates when a new level of knowledge has been attained.
- Guess State (G): represents the state when a guess is made and is the end of any epoch. Guess states are therefore "absorbing" states.

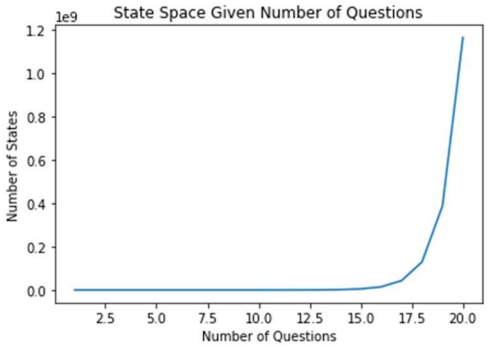
The game terminates when one animal is guessed. (only one-shot allowed)

	Code Index					State Transition Function		
No	Rnd	Q1	Q2	Guess	State Type	Current State S	Next State S+1	Comment
1	0	0	0	0	K	0000	{0001, 0002, 1100, 1200}	Ask a Q, or guess
2	0	0	0	1	G	0001		Guess Wrong
3	0	0	0	2	G	0002		Guess Right
4	1	1	0	0	K	1100	{1101, 1102, 2110, 2120}	Ask a Q or guess
5	1	2	0	0	K	1200	{1201, 1202, 2210, 2220}	Ask a Q or guess
6	1	1	0	1	G	1101		Guess Wrong
7	1	2	0	1	G	1201		Guess Wrong
8	1	1	0	2	G	1102		Guess Right
9	1	2	0	2	G	1202		Guess Right
10	1	0	1	0	K	1010	{1011, 1012, 2110, 2210}	Ask a Q or guess
11	1	0	2	0	K	1020	{1021, 1022, 2120, 2220}	Ask a Q or guess
12	1	0	1	1	G	1011		Guess Wrong
13	1	0	2	1	G	1021		Guess Wrong
14	1	0	1	2	G	1012		Guess Right
15	1	0	2	2	G	1022		Guess Right
16	2	1	2	0	K	2120	{2121, 2122}	Final Round, must guess
17	2	1	1	0	K	2110	{2111, 2112}	Final Round, must guess
18	2	2	1	0	K	2210	{2211, 2212}	Final Round, must guess
19	2	2	2	0	K	2220	{2221, 2222}	Final Round, must guess
20	2	1	2	1	G	2121		Guess Wrong
21	2	1	1	1	G	2111		Guess Wrong
22	2	2	1	1	G	2211		Guess Wrong
23	2	2	2	1	G	2221		Guess Wrong
24	2	1	1	2	G	2112		Guess Right
25	2	1	2	2	G	2122		Guess Right
26	2	2	1	2	G	2212		Guess Right
27	2	2	2	2	G	2222		Guess Right

2.1.6 Scaling the State & Action Space

A feature of the 20 Question game type is that the state space increases exponentially with the number of questions, therefore this state representation is likely not optimal for environments with large numbers of possible questions.

The action space is the number of questions the agent can ask in addition to guessing any of the animals and also scaled exponentially. In order to reduce the search space, we restrict the agent to not being able to ask the same question twice.

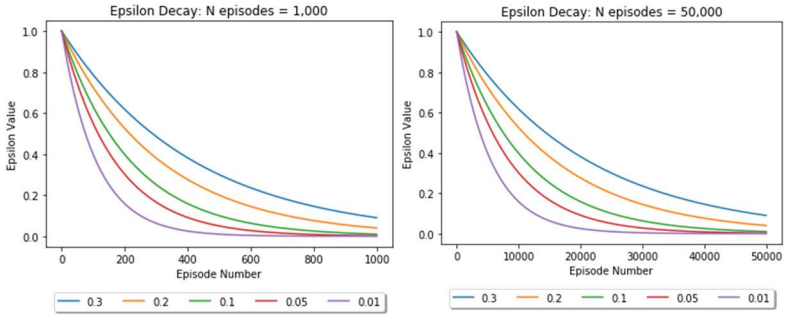


3 Q-Learning Policy

The agent's policy dictates the optimal action it should take for a given state to maximize expected return. We have decided to use an E-greedy policy in which the agent takes a random action with probability epsilon (ε) and follows the greedy action with probability 1- ε. The value of ε determines the tradeoff between exploration: trying different possible actions for a given state and exploitation: taking the actions which have the highest expected Q-values based on prior episodes.

3.1 Using Epsilon-midpoint for Epsilon Decay

We also use an epsilon decay function to capture the notion that the agent requires less exploration as it learns more about its environment.



A common method for decaying the epsilon (ε) is to multiply it each episode by a discount factor ("df") where 0 < df < 1. A limitation of this approach for grid-searching is that very small changes in df could have a significant effect on ε over fixed episodes. As this is a challenging value to interpret, we defined the desired epsilon value at the middle episode and solved for the implied decay factor (df) in order to create an epsilon decay function which is invariant to the number of episodes.

3.2 Early Stopping

To terminate training the agent, we use early stopping where the agent is assumed to have converged rather than terminating at an arbitrary number of episodes. We use variance as a proxy for convergence and stop training once the variance of the return from the last 100 episodes of the agent's greedy actions falls below a threshold. The lower the threshold the longer the agent takes to converge and the more stable the predictions (all else equal). The optimal value of the variance threshold is a function of the expected return for a given environment.

4 Parameter Values for Q-Learning

The Q-learning algorithm is a Temporal Difference Learning method which is a balance between the Monte Carlo method and the Dynamic Programming approach. Like the MC, TD does not assume any prior knowledge of the environment and like DP, bootstraps by updating estimates based on other estimates.

The Q-learning algorithm will update an estimated value/reward for a state based on estimates for succeeding states (the estimates of these succeeding states having been built up during previous episodes). Initially, with no prior knowledge, the agent will have no estimates, but gain rewards (positive or negative) as it takes an action to transition from state to state.

The value update formula is as follows: [6]

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

A Description of Q-learning parameters is described in table below:

Parameter Symbol	Parameter Name	Typical Value Range	Value in Small game example	Commentary
α	alpha	0 - 1	0.5	Learning rate
γ	gamma	0 - 1	0.8	Discount factor, if 0 myopic only immediate reward matters. If 1 all rewards are equally weighted
ε	epsilon	0 - 1		Probability of select to take a random action, vs (1-ep) being probabbility of selecting the action with highest value v.
df		0 - 0.01	0.01	Tradition decay factor for epsilon, explore early on and exploit later as knowledge gained. Not used in our experiment.
ε-mid-point		0 - 0.3	0.01 - 03	Manufactured decay parameter for these experiments.
Other terms				
r				Immediate Reward rec'd moving to next state / signals how good or bad a state is
R				Accumulated discounted reward or return
π	pi			Policy. Epsilon greedy policy applied in this experiment. Ie Agent mostly picks the current best option ("greedy"), but sometimes agent picks a random option with a small (epsilon) probability.
	Epochs			No of learning epochs /episodes
	Samples		5	No of samples per parameter config to average. 5 selected due to time constraints of project.

5 Updating the Q-Matrix

5.1 R-Matrix for Small Game

As the Q-matrix consists of the Q-values which will converge on the estimated return values and are therefore related to the reward matrix we first define the R-matrix. The reward table is only one example as in this game 3 similar grids exist (1 for each animal). In effect "animals" are an extra dimension for the grid. For simplicity we are only showing the grid for a Swan.

R Matrix Example – Swan Grid. *The R matrix is transposed to fit on the page.*

			STATES (G=Terminating State)																											
			K	G	G	K	K	G	G	G	G	K	K	G	G	G	G	K	K	K	K	G	G	G	G	G	G	G	G
			0000	0001	0002	1100	1200	1101	1201	1102	1202	1010	1020	1011	1021	1012	1022	2120	2110	2210	2220	2121	2111	2211	2221	2112	2122	2212	2222
Actions	AskQ	Aquatic	A	-1	-	-	-	-	-	-	-	-	-	-1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	AskQ	Airborne	B	-1	-	-	-	-1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Guess	Swan	C	5	-	5	-	5	-	-	-	5	-	5	-	-	-	5	-	-	-	5	-	-	-	-	-	-	-	5
	Guess	Dolphin	D	-10	-10	-	-	-10	-	-10	-	-	-	-10	-	-10	-	-	-	-	-	-10	-	-	-	-10	-	-	-	-
	Guess	Fruitbat	E	-10	-10	-	-	-10	-	-10	-	-	-	-10	-	-10	-	-	-	-	-	-10	-	-	-	-10	-	-	-	-
	Guess	Cavy	F	-10	-10	-	-	-10	-	-10	-	-	-	-10	-	-10	-	-	-	-	-	-10	-	-	-	-10	-	-	-	-

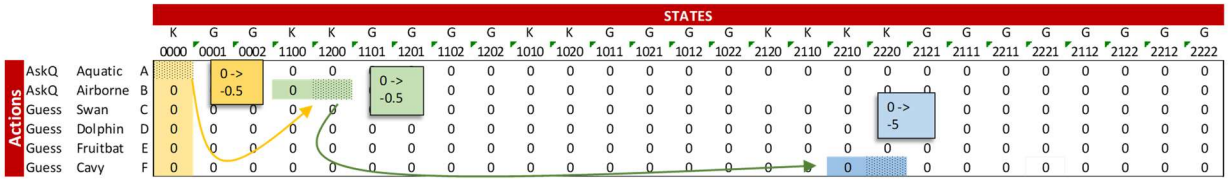
Note: "Guess" actions lead to terminal states which are absorbing (highlighted in blue). Certain states are not available for the Swan R-Matrix which are a possibility on the grid for another animal e.g. states where the answer to either question is false.

5.2 Updating the Q-Matrix Example

The following section describes how the Q-learning matrix is updated. This example covers the steps in initializing the Q-matrix and completing one episode to show how estimates are made in the very first episode of the game for the small environment.

A-Initialize parameters: alpha, gamma, epsilon, initial state= 0000, the goal state is any "guess" state. Reward function: guess right = +5, guess wrong = -10, ask question =-1.

B-Initialize Matrix: Initialize Q matrix with zeros.



C=1st Episode 1st Action

Compare ε to random number ("Beta"). Assume Beta< ε so agent makes random choice for next step.
Chooses to ask Aquatic question and answer is "Yes" so next state is 1200. Update calculation is:

$Q_{new}(0000, A) = Q_{old}(0000,A) + \alpha[(rew(0000,A) + \gamma. \max Q(\text{next states from } 0000, \text{consider all actions}=0)) - Q_{old}(0000,A)]$
$Q_{new}(0000, A) = 0 + 0.5[(-1 + 0.8. \max Q(0)) - 0] = -0.5$
So $Q_{new}(0000, A)$ in Q matrix is updated from 0 to -0.5

D=1st Episode 2nd Action

At state 1200 (after asking aquatic question and getting "Yes"). Compare ε to random number. Assume Beta< ε so agent makes random choice for next step. Chooses to ask Airborne question and answer is "Yes" so next state is 2220. Update calculation is:

$Q_{new}(1200, B) = Q_{old}(1200, B) + \alpha[(rew(1200,B) + \gamma. \max Q(\text{next states.from1200, all actions}=0)) - Q_{old}(1200,B)]$
$Q_{new}(1200, A) = 0 + 0.5[(-1 + 0.8. \max Q(0, 0)) - 0] = - 0.5$
So $Q_{new}(1200, A)$ in Q matrix is updated from 0 to -0.5

E=1st Episode 3rd Action

At state 2220. Only available action is for agent to make a guess. Agent guesses Cavy which is wrong (not Swan).
Update calculation is:

$Q_{new}(2220, F) = Q_{old}(2220,B) + \alpha[(rew(2220,F) + \gamma. \max Q(\text{next states.from2220, all actions})) - Q_{old}(2220,F)]$
$Q_{new}(2220, F) = 0 + 0.5[(-10 + 0.8. \max Q(0, 0)) - 0] = -5$
So $Q_{new}(2200, F)$ in Q matrix is updated from 0 to -5

The episode is complete as the agent is in a terminal state.

6 Experimental Results

In order to determine the optimal hyperparameters for the Q-learning model, we run a grid search across the key parameters: alpha, gamma, and epsilon-decay midpoint (df) across two reward functions. (Full results contained in appendix)

As we are using an epsilon-decay policy in both models, the results have variability given the random exploration caused by epsilon. In order to better determine whether a given hyperparameter configuration results in improved performance, we averaged the results of five trained models.

The key performance metric we are comparing is the average number of questions assuming the model achieves a given level of accuracy. The model will be applied in a domain where accuracy is more important than number of questions therefore, we set the accuracy threshold at 95%, then consider the average number of questions, and finally we consider the number of episodes the model takes before convergence.

6.1 Findings from Gridsearch using the Baseline Reward Function

The model was run for each gridsearch combination until the lower of convergence or 20,000 episodes due to computational constraints. The baseline Reward Function we applied was as follows: guessing an animal correctly =+5, guessing incorrectly =-10, and asking a question: -1. We examine the impact of varying the key hyperparameters below. Please note that "Below threshold" indicates the % of episodes that did not achieve an accuracy of 95%.

Alpha - Learning Rate (theory: a higher alpha causes faster update towards new estimate from old)
On average an alpha of 0.20 produced the best results in terms of meeting threshold accuracy (92% of trials) and lowest episodes to convergence. The average question number was highest at 5.27 with the lowest question number being 5.07 (but with 3 times more runs not passing the accuracy threshold). Intuitively we expected alpha to have a greater value although one common strategy is to use a decay for alpha as the model confidence increases in its estimate.

Impact of Varying Alpha

Value	Question Number	Below Thresh	Episode #
0.05	5.26	28.0%	15,696
0.20	5.27	8.0%	13,880
0.50	5.19	16.0%	13,989
0.80	5.13	36.0%	15,389
0.95	5.07	24.0%	16,236

Gamma - Discount Rate (theory: if gamma near 0, immediate reward is preferred, if 1 all rewards have same weight).
The gridsearch finding shows that a higher gamma was most successful wrt accuracy. Also, at low values a significant % of episodes did not get above the 95% accuracy threshold.
This intuitively makes sense given the reward structure of negative rewards for asking questions and guessing wrongly and only a positive reward at the end of an episode for a correct guess. The agent does better when gamma allows consideration of prospective future positive rewards.

Impact of Varying Gamma

Value	Question Number	Below Thresh	Episode #
0.05	5.19	44.0%	16,468
0.20	5.18	32.0%	16,450
0.50	5.23	24.0%	15,497
0.80	5.18	12.0%	13,553
0.95	5.17	0.0%	13,897

Epsilon Mid-point (theory: a lower E-mid-point value produces a sharper Epsilon curve meaning a faster switch from predominantly exploration episodes towards exploitation episodes)
The gridsearch showed that a lower epsilon midpoint was best for accuracy, episodes to convergence but had little impact wrt to average question numbers. Intuitively, a near-zero value would cause the agent to quickly pursue an exploitation policy so an optimal value between 0.01 and 0 may exist. At highest value of 0.3, convergence was never been attained by 20,000 epochs.

Impact of Varying Epsilon-Midpoint

Value	Question Number	Below Thresh	Episode #
0.01	5.18	0.0%	11,950
0.05	5.18	8.0%	13,568
0.10	5.18	16.0%	15,162
0.20	5.21	28.0%	17,903
0.30	5.22	60.0%	19,875

6.2 Changing the Reward Function to Binary

The gridsearch was repeated using an alternative reward function as follows: guessing an animal correctly = +10, asking a question = 0, guessing an animal wrong = 0. Binary reward functions are common in many environments to avoid reward shaping although as the agent has no signal to optimize for questions asked, we hypothesized that this change in rewards should increase episodes/epochs required for convergence.

The impact of each parameter is summarized below (Note that the Accuracy % is the average accuracy attained rather than the "% below 95% threshold" metric used previously).

Impact of Varying Epsilon

Value	Question Number	Accuracy
0.01	2.29	27%
0.05	2.25	26%
0.10	2.34	27%
0.20	2.36	29%
0.30	2.21	26%

Impact of Varying Gamma

Value	Question Number	Accuracy
0.05	1.77	21%
0.20	2.15	25%
0.50	2.37	28%
0.80	2.53	30%
0.95	2.63	31%

Impact of Varying Alpha

Value	Question Number	Accuracy
0.05	1.05	12%
0.20	2.24	25%
0.50	2.67	32%
0.80	2.74	33%
0.95	2.76	34%

With the alternative gridsearch no combination got near the previous 95% threshold we had set. The maximum level of average accuracy attained was 34%. The low average question numbers are not valuable metrics given the inaccuracy. In no hyperparameter combination did convergence occur before 20,000 episodes. Given the poor results, comments on the hyperparameters may be less valuable, but we note that relatively, accuracy improved with higher values alpha and gamma and declined for the epsilon midpoint.

The results from our analysis confirmed our intuition that modifying the reward function would have a dramatic impact on agent performance.

6.3 Detailed Comparison of Three Models

Following the gridsearch exercise we identified three models to compare in greater depth. The first model applies the optimal hyperparameters based on our gridsearch to the base reward function and the second applies parameters which perform poorly. The third model applies the same parameters as the first although with the alternate binary reward function (i.e. the agent receives a +10 reward for correctly guessing the animal and no other feedback).

Base Reward Function Model (Best Hyperparameters)

Model Hyperparameters

Metric	Value
Alpha	0.80
Gamma	0.95
Epsilon Midpoint	0.01
Reward Function	(+5, -10, -1)

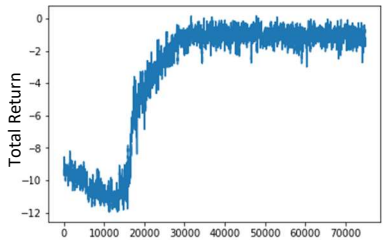
Performance Metrics

Metric	Value
Accuracy	100%
Average Number of Questions	4.84
Episode Convergence	35,000

Question Decision Path

	Animal	Q_Nodes	Predicted	N_Questions
0	aardvark	[Aquatic, Predator, Eggs, Backbone]	-	4
1	cavy	[Aquatic, Predator, Nlegs_5, Airborne, Eggs]	-	5
2	chicken	[Aquatic, Predator, Nlegs_5, Airborne, Eggs, Backbone]	-	6
3	clam	[Aquatic, Predator, Eggs, Airborne, Backbone]	-	5
4	crab	[Aquatic, Predator, Airborne, Eggs, Nlegs_5, Backbone]	-	6
5	dolphin	[Aquatic, Predator, Airborne, Eggs]	-	4
6	fruitbat	[Aquatic, Predator, Nlegs_5, Airborne, Eggs]	-	5
7	gull	[Aquatic, Predator, Airborne]	-	3
8	haddock	[Aquatic, Predator, Airborne]	-	3
9	kiwi	[Aquatic, Predator, Eggs, Airborne, Backbone]	-	5
10	ladybird	[Aquatic, Predator, Eggs, Airborne, Backbone]	-	5
11	ostrich	[Aquatic, Predator, Nlegs_5, Airborne, Eggs, Backbone]	-	6
12	penguin	[Aquatic, Predator, Airborne, Eggs, Nlegs_5, Backbone]	-	6
13	scorpion	[Aquatic, Predator, Eggs, Backbone]	-	4
14	starfish	[Aquatic, Predator, Airborne, Eggs, Nlegs_5]	-	5
15	swan	[Aquatic, Predator, Airborne]	-	3
16	vulture	[Aquatic, Predator, Eggs, Airborne, Backbone]	-	5
17	wasp	[Aquatic, Predator, Nlegs_5, Airborne, Eggs, Backbone]	-	6
18	worm	[Aquatic, Predator, Nlegs_5, Airborne, Eggs, Backbone]	-	6

Performance Over Episodes



Base Reward Function Model (Worst Hyperparameters)

Model Hyperparameters

Metric	Value
Alpha	0.50
Gamma	0.05
Epsilon Midpoint	0.30
Reward Function	(+5, -10, -1)

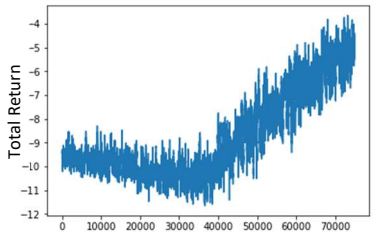
Performance Metrics

Metric	Value
Accuracy	100%
Average Number of Questions	5.21
Episode Convergence	N/A

Question Decision Path

	Animal	Q_Nodes	Predicted	N_Questions
0	aardvark	[Nlegs_5, Backbone, Aquatic, Airborne, Eggs, Predator]	-	6
1	cavy	[Nlegs_5, Backbone, Aquatic, Predator, Eggs, Airborne]	-	6
2	chicken	[Nlegs_5, Airborne, Eggs, Backbone, Predator, Aquatic]	-	6
3	clam	[Nlegs_5, Airborne, Predator, Aquatic, Eggs, Backbone]	-	6
4	crab	[Nlegs_5, Backbone, Aquatic]	-	3
5	dolphin	[Nlegs_5, Eggs, Airborne, Backbone, Predator, Aquatic]	-	6
6	fruitbat	[Nlegs_5, Eggs, Airborne]	-	3
7	gull	[Nlegs_5, Backbone, Airborne, Eggs, Predator, Aquatic]	-	6
8	haddock	[Nlegs_5, Airborne, Predator, Aquatic]	-	4
9	kiwi	[Nlegs_5, Aquatic, Eggs, Predator, Airborne, Backbone]	-	6
10	ladybird	[Nlegs_5, Aquatic, Backbone, Eggs, Predator, Airborne]	-	6
11	ostrich	[Nlegs_5, Predator, Aquatic, Eggs, Airborne, Backbone]	-	6
12	penguin	[Nlegs_5, Predator, Aquatic, Airborne, Eggs, Backbone]	-	6
13	scorpion	[Nlegs_5, Eggs, Airborne, Backbone]	-	4
14	starfish	[Nlegs_5]	-	1
15	swan	[Nlegs_5, Airborne, Eggs, Backbone, Aquatic, Predator]	-	6
16	vulture	[Nlegs_5, Aquatic, Backbone, Airborne, Eggs, Predator]	-	6
17	wasp	[Nlegs_5, Predator, Aquatic, Eggs, Airborne, Backbone]	-	6
18	worm	[Nlegs_5, Aquatic, Airborne, Eggs, Backbone, Predator]	-	6

Performance Over Episodes



Binary Reward Function Model (Best Hyperparameters)

Model Hyperparameters

Metric	Value
Alpha	0.80
Gamma	0.95
Epsilon Midpoint	0.01
Reward Function	(+10, 0, 0)

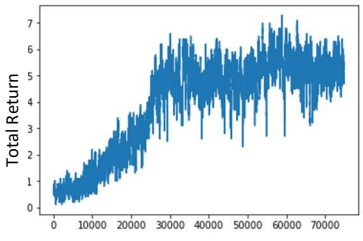
Performance Metrics

Metric	Value
Accuracy	**53%**
Average Number of Questions	3.21
Episode Convergence	40,000

Question Decision Path

	Animal	Q_Nodes	Predicted	N_Questions
0	aardvark	[Aquatic, Backbone, Predator, Airborne, Eggs]	-	5
1	cavy	[Aquatic, Backbone, Predator, Eggs, Airborne]	-	5
2	chicken	[Aquatic, Backbone, Predator, Eggs]	-	4
3	clam	[Aquatic, Backbone, Airborne]	worm	3
4	crab	[Aquatic, Backbone]	starfish	2
5	dolphin	[Aquatic, Backbone]	-	2
6	fruitbat	[Aquatic, Backbone, Predator, Eggs, Airborne]	-	5
7	gull	[Aquatic, Backbone]	dolphin	2
8	haddock	[Aquatic, Backbone]	dolphin	2
9	kiwi	[Aquatic, Backbone, Predator, Airborne, Eggs]	-	5
10	ladybird	[Aquatic, Backbone, Airborne]	wasp	3
11	ostrich	[Aquatic, Backbone, Predator, Eggs]	chicken	4
12	penguin	[Aquatic, Backbone]	dolphin	2
13	scorpion	[Aquatic, Backbone, Airborne]	worm	3
14	starfish	[Aquatic, Backbone]	-	2
15	swan	[Aquatic, Backbone]	dolphin	2
16	vulture	[Aquatic, Backbone, Predator, Airborne]	-	4
17	wasp	[Aquatic, Backbone, Airborne]	-	3
18	worm	[Aquatic, Backbone, Airborne]	-	3

Performance Over Episodes



6.4 Performance Over Episodes

The first model performs the best as it achieved 100% accuracy while asking only 4.84 questions on average before guessing and visually converges after only ~35,000 episodes (see graphs). This contrasts with the second model which asked 5.21 questions on

average and did not visually converge during the 75,000 episodes. It should be noted that this model still achieved 100% accuracy despite requiring more questions on average in contrast with the binary reward function which achieved only 53% accuracy and appeared to stop improving after 40,000 episodes.

The "Performance Over Episodes" graphs indicate when convergence has been attained (or not) however the total return values can be compared within graphs but not across the across graphs e.g. the binary reward model has the highest total return but this is due to the reward profile having no negative penalty.

6.5 Analyzing the Decision Path

The decision tree nodes show how, given the rewards and hyperparameters the agent converged on the optimal set of questions to ask given the answers from the respondent.

Both the "Best Hyperparameter" runs identified "Is the animal aquatic?" as the optimal first question to ask. This makes intuitive sense as this question divides the 19 animals into 2 groups of 7 and 12, (in fact only the "predator" question is more efficient splitting the group into 11 and 8.

By contrast, the "Worst Hyperparameter" run converged on asking as its 1st question: "Does the animal have 5 legs?" which divides the group less efficiently into 18 and 1 (a starfish). We speculated that in early episodes the agent randomly selected the 5-leg question and then randomly chose to guess starfish straight after.

The 3 case study review results confirm the findings that the reward function is a more significant determinant of model performance than the hyperparameter configuration.

Decision Tree | Question Decision Path

	Animal	DT_Nodes	Predicted	N_Questions
0	aardvark	[Predator, Aquatic, Backbone, Airborne, Eggs]	-	5
1	cavy	[Predator, Airborne, Aquatic, Eggs]	-	4
2	chicken	[Predator, Airborne, Backbone, Eggs, Aquatic]	-	5
3	clam	[Predator, Aquatic, Backbone, Eggs, Airborne]	-	5
4	crab	[Predator, Aquatic, Backbone, Nlegs_5]	-	4
5	dolphin	[Predator, Aquatic, Backbone, Airborne, Eggs]	-	5
6	fruitbat	[Predator, Airborne, Backbone, Eggs]	-	4
7	gull	[Predator, Aquatic, Backbone, Airborne]	-	4
8	haddock	[Predator, Airborne, Aquatic]	-	3
9	kiwi	[Predator, Aquatic, Backbone, Airborne, Eggs]	-	5
10	ladybird	[Predator, Aquatic, Backbone, Eggs, Airborne]	-	5
11	ostrich	[Predator, Airborne, Aquatic, Eggs, Backbone]	-	5
12	penguin	[Predator, Aquatic, Backbone, Airborne, Eggs]	-	5
13	scorpion	[Predator, Aquatic, Backbone, Eggs]	-	4
14	starfish	[Predator, Aquatic, Backbone, Nlegs_5]	-	4
15	swan	[Predator, Airborne, Backbone, Eggs, Aquatic]	-	5
16	vulture	[Predator, Aquatic, Backbone, Airborne]	-	4
17	wasp	[Predator, Airborne, Backbone]	-	3
18	worm	[Predator, Airborne, Aquatic, Eggs, Backbone]	-	5

6.6 Comparison of Performance to Decision Trees

A decision tree model by comparison achieved 100% accuracy with only 4.42 questions. This is to be expected as it uses the knowledge base and splits based on entropy. A reinforcement learning approach has multiple advantages over decision trees including: not requiring a knowledge base, robustness to noisy answers (mistaken answers by the respondent), and an ability to easily optimize the decision path to certain animals through changing the reward function

6.7 Additional Experiments

6.7.1 Full Exploration vs Full Exploitation

We experimented with running 1,000,000 episodes with only exploration to see if the agent could find an optimal policy. We also experimented with 75,000 episodes of only exploitation.

- The agent was unable to find an optimal policy with only random exploration and achieved only 21% accuracy
- Randomly selecting the animal at the beginning of each episode adds some stochasticity to the problem which allows the agent to achieve 100% accuracy in 5.11 questions using only exploitation. Given the agent does not spend time exploring it is able to converge in far fewer episodes (~5,000). However, the decision path shows far more variation.

6.7.2 Alternative Reward Functions

Using the optimal hyperparameters we tried other reward functions with the following results (one run of 75,000 episodes each):

- Pro-risk Reward Strategy (correct guess: +10, incorrect guess: -5, ask question: -1): Achieved performance very similar to base case with 100% accuracy with 5.11 questions and visual convergence after 35,000 episodes.
- Balanced Low Question Cost Strategy (+50, -50, -1): Also achieved performance similar to base case with 100% accuracy although slightly more questions at 5.26 likely reflecting the reduced question cost.
- Punitive Strategy (0, 0, -10): achieved 100% accuracy although at 5.63 questions which is noticeably worse.

7 Advanced Reinforcement Learning Techniques: Deep Q-Learning and Defining a Bayesian State Space

One of the major constraints of tabular Q-learning is that it can only be used to solve problems which have state-action spaces that are small enough to fit into memory. In environments with large or infinite state-action spaces we need to use function approximation to learn a mapping between a state-action pair and its associated q-value. As discussed in the environment definition, our state space grows exponentially with the number of questions therefore we need to use function approximation and have implemented a Deep-Q network.

Vanilla Deep-Q networks showed poor performance with very unstable convergence properties and long training times. This is consistent with other research[7] and therefore we implemented two additional variants to improve performance:

- **Experience replay:** rather than training the network online as the (state, action, reward, next state) pairs occur, we define a memory bank of size 5,000 which the pairs are stored in. Each step we then randomly sample a batch of 32 pairs and train the network. This helps the network reduce overfitting to recent experience thereby smoothing the training process. It also removes the correlation between samples which improves efficiency.[8]
- **Double Q-learning (DDQN):** Deep-Q networks attempt to predict the Q value for a given state action pair which is based on the maximum q-value of the next state which itself is a prediction. This makes the network very volatile as both values are moving. Additionally, in some environments the network overestimates the action values. One solution to this is to use two networks: one to select the best action and one to calculate the target Q value.

The neural network architecture used is relatively shallow given the relatively low dimensionality of the state space and consists of one layer of 528 nodes and another of 256. After training the model for 13 hours and ~400k episodes, the performance improved noticeably although the model still required a long training time and got stuck in local maxima.

The model appears very risk adverse and attempts to find the lowest number of questions it could ask sequentially and consistently guess each animal. This is not the desired performance as the model does capture the information contained in the Respondent's replies to questions. This is likely due to the reward function although we were unable to find a reward function which achieved optimal behavior without significant reward shaping. Smaller environments also show the same performance characteristics.

Decision Path for Complex Environment

	Animal	Q_Nodes	Predicted	N_Questions
0	aardvark	['Backbone', 'Eggs', 'Nlegs_5', 'Airborne', 'Predator', 'Aquatic']	-	6
1	cavy	['Backbone', 'Eggs', 'Nlegs_5', 'Airborne', 'Predator', 'Aquatic']	-	6
2	chicken	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
3	clam	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Aquatic', 'Predator']	-	6
4	crab	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Aquatic', 'Predator']	-	6
5	dolphin	['Backbone', 'Eggs', 'Nlegs_5', 'Airborne', 'Predator', 'Aquatic']	-	6
6	fruitbat	['Backbone', 'Eggs', 'Nlegs_5', 'Airborne', 'Predator', 'Aquatic']	-	6
7	gull	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
8	haddock	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
9	kiwi	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
10	ladybird	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Predator']	-	5
11	ostrich	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
12	penguin	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
13	scorpion	['Backbone', 'Nlegs_5', 'Eggs', 'Aquatic', 'Airborne', 'Predator']	-	6
14	starfish	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Predator', 'Aquatic']	-	6
15	swan	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
16	vulture	['Backbone', 'Eggs', 'Airborne', 'Nlegs_5', 'Predator', 'Aquatic']	-	6
17	wasp	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Predator', 'Aquatic']	-	6
18	worm	['Backbone', 'Nlegs_5', 'Eggs', 'Airborne', 'Aquatic', 'Predator']	scorpion	6

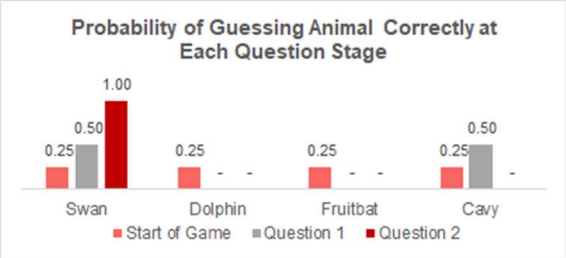
Decision Path for Smaller Environment

	Animal	Q_Nodes	Predicted	N_Questions
0	antelope	[Catsize, Tail, Eggs, Aquatic]	-	4
1	bass	[Catsize, Tail, Eggs, Aquatic]	-	4
2	bear	[Catsize, Tail, Eggs, Aquatic]	-	4
3	cavy	[Catsize, Tail, Eggs, Aquatic]	-	4
4	dolphin	[Catsize, Tail, Eggs, Aquatic]	-	4
5	fruitbat	[Catsize, Tail, Eggs, Aquatic]	-	4
6	rhea	[Catsize, Tail, Eggs, Aquatic]	-	4
7	swan	[Catsize, Tail, Eggs, Aquatic]	-	4

7.1 Reformulating the Environment

Another approach designed to help the Deep-Q agent converge more quickly is providing the agent with the knowledge base. This can be done by reformulating the state space as the probability distribution over possible animals and the action space as asking a question or guessing the animal with the highest probability (in order to reduce the size of the action space). We also include the question number in the state space so the agent knows when the game will end. We train the same Deep-Q network defined above.

The state space now becomes continuous and updates given the knowledge base. This also allows the agent to use a bayesian approach through including a non-uniform prior to encompass information about the prior distribution. For example, if over many episodes the agent learns that swans are chosen by the respondent 40% of the time rather than initializing the probability at 25% it would be 40% and allow the agent to learn to guess swan more quickly.



The performance was also very underwhelming with the agent seemingly not incorporating the knowledge base in its decision making and the model never achieving convergence. We believe that the poor performance in both Deep-Q Learning algorithms is due to the incorrect reward function. We tried many varied reward functions including a decaying reward by a function of steps and all the reward functions noted above without success. Another possibility is the large action space resulted in the agent failing to explore optimal state-action pairs.[9]

8 Parallelisms Between Q-learning and Rescorla-Wagner

The relationship between reinforcement learning and associative learning is well established.[10] [11] [5] The goal of associative learning is to understand the mechanism by which humans and animals learn to relate to events in their environment.[12] In 1972 the Rescorla-Wagner model of associative learning, an error correction model, proposed that cumulative learning, "V", the weight of a specific link between a given CS (conditioned stimulus) and a US (unconditioned stimulus) related to the previous prediction (value of previous V) plus an adjustment for learning from the latest experience / trial.[6]

The RW formula is: Error term: actual outcome less prediction of outcome

Prediction $V_i^n = V_i^{n-1} + \alpha_i \cdot \beta_{US} (\lambda^n - \sum^z_{i=A} V_i^{n-1} \cdot X_i^n)$

α_i = learning rate of I, β_{US} = learning (associability) rate of US
λ^n=actual outcome (maximum conditioning possible for the US), $-\sum^z_{i=A} V_i^{n-1} \cdot X_i^n$ = sum of all present predictors of same outcome, total previous reward.
X_i^n=stimulus presence 1 or 0

In reinforcement learning, temporal difference learning methods (TDs) do not require prior knowledge of the environment and "bootstrap" i.e. update estimates based on other estimates. The Q-learning algorithm is a TD(0) method and has the following update rule:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Where: a=action, s=state, α =learning rate. γ= Discount factor, r = Immediate reward rec'd moving to next state

A comparison of the Q-learning update rule with the RW formula reveals several parallelisms:

1. New values or estimates are based on the previous value or estimate plus an adjustment (In the Q-Learning model the delta is called the "TD Error signal").
2. In the RW model a component of the adjustment is the "error term" and the Q-learning equivalent is the "learned value" less the "old value".
3. The adjustment is proportional to a learning rate variable which has a value range between 0 and 1 and represents the "learning step" size.
4. An innovation of the RW model was to move from a "local error" difference based on outcome λ and V_i^{n-1}(the value from the last similar trial used by previous models) to a "global error" based on outcome λ and $\sum^z_{i=A} V_i^{n-1}$ being the sum of the

value all previous outcomes. In Q-learning the Q value for the current state S_t captures all information from the past history (Markov Decision process assumption).

5. Both V and Q-Learning curves are asymptotic, and they will converge towards a final value.
6. The vanilla RW model described learning in terms of trials and Q-Learning describes updates in terms of states and actions. Neither model formulae build-in the impact of chronological time. (The RW model does consider "context" with inter-trial intervals).
7. Temporal Difference is a real-time version of the RW model with the delta update adjusted by an eligibility trace function. Q-learning, another TD model, has versions including eligibility traces e.g. Peng's Q(λ) algorithm.

Two differences between the models are as follows:

1. The Q-Learning reward is expressed as a single utility measure whilst the reward in RW can vary quantitatively but also qualitatively (e.g. combinations food and / or water and / or shocks are different stimuli).
2. The RW model can model compound stimuli using a "summation assumption".

Sutton, in his 1988 "Temporal Difference Learning" paper, suggests that animals demonstrating Pavlovian or classical conditioning "may be using a TD method".[13]

9 Implementing Error Correction Models as Deep Reinforcement Learning Architectures

The original Rescorla-Wagner model was an advance on previous models by replacing a "local error" difference based on actual outcome λ and V_i^{n-1} to a "global error" based on outcome λ and $\sum^{z}_{i=A} V_i^{n-1}$. RW is relatively simple and makes successful predictions. However, there are some more complex effects that are not explained by RW.

The Temporal Difference model has been described as a "real time RW model" as it considers updates over time (including inter-stimuli intervals). This has 3 enhancements:

- It allows new more realistic ways for a stimulus to be represented. For instance, a stimulus may have several components with varying intensities over time or may be presented sequentially.
- The TD compares actual outcome λ with the total change in outcome prediction between the 2 most recent predictions.
- The delta adjustment is now multiplied by an eligibility trace component so when a component stimulus ends the effect decays.

The TD model has been applied to neuroscience questions and has successfully modelled predictive reward signals in dopamine neurons.[14] However, there are several phenomena that have not been explained by associated learning algorithms. Examples include learning about now "absent" stimuli or evidence of learning between neutral stimuli questions.[14] We attempted to use the Simultaneous and Serial Configural-cue Compound Stimuli Temporal Difference model to model a very simple instance of our 20 Questions game for comparative purposes (contained in the appendix).

Any implementation of a "error correction" model would require the following features:

- Accept multiple stimulus inputs over time and to treat them as single stimuli and/or compound stimuli
    - Note: it's not necessarily a requirement to receive inputs in real-time
- Accept states of inter-stimuli intervals e.g. to simulate context or context changes
- Accept multiple different stimuli types
- Accept quantitatively and qualitatively different unconditioned stimuli
- Compute "global error" terms
- Ability to model a memory and decay of memory

Based on the above requirements the most likely architecture that can manage these features is a neural network with multiple layers between input and output layer i.e. a deep neural network which has already been successful in replicating sophisticated tasks such as image recognition that humans find easy.[15] This also makes intuitive sense: that an objective is to predict complex responses of human /animal behavior could be based on a model inspired by biological neural networks.

As TD models are designed to be able to capture temporal relationships, the recurrent neural network family of models would likely be the best architecture to capture sequential patterns. Possible architectures include vanilla RNNs, Long-Short Term Memory networks, Gated Recurrent Units, or other designs. Of course this depends on how the state space is represented as Convolutional Neural Networks are state of the art at extracting features in high dimensional image data which does not have a sequential aspect.

Deep Q-networks are currently state of the art in many deep reinforcement learning tasks and are flexible in that they can have any network architecture in the prediction layer so may be the best candidate for modelling associative learning tasks.

In terms of the practical implementation of real data gathering, techniques have advanced greatly since the 1890's when Pavlov rang bells and measured salvia from test tubes inserted into a dog's cheeks. The forms of conditioned stimulus being tested need to be sophisticated and controlled to test sequences of stimuli that may be compound, serial and/or varying over time and that are tested within defined contexts. These stimuli profiles must also be repeatable. So, stimuli are largely electronic actuators such as lights and tones whose parameters/profiles are controlled by computer applications. This also facilitates comparisons to computer models.

10 Proposed Future Work

As part of future work we suggest applying other deep reinforcement learning algorithms and policy based methods and training for a longer duration. Even with the improvements made to the Deep-Q model it was unable to produce results of value.

One of the main advantages of a reinforcement learning approach to 20 Questions (aside from not requiring a knowledge base) is that it should be robust to noise. Therefore, it would be interesting to determine how adding noise to the environment impacts performance. This could be done through the Respondent providing incorrect feedback.

Another area to explore is a further gridsearch of the hyperparameters and reward structure including running the gridsearch for a longer duration and with more trials to reduce variance in the results.

11 Conclusion

We were able to represent the classic 20-Questions game as reinforcement learning task solvable by the tabular Q-Learning algorithm. While the agent never outperformed the decision tree it performed closely achieving 100% accuracy with an average 4.84 questions versus 4.42 questions and was quite robust to different hyperparameters under the correct reward function. Using a grid-search exercise we identified optimal parameters for alpha, gamma, and our epsilon-midpoint variant of epsilon. It was also noted that the reward function had a very significant effect on episodes to convergence in the binary case.

The performance of the Deep-Q Network under both the default environment and the reformulated Bayesian state space was underwhelming with neither model achieving strong performance and the Bayesian reformulation not converging. We have highlighted this as an area for future work. We also discussed the relationship between Q-learning and error-correction models of associative learning and how deep learning models can be applied to associative learning.

Appendix A: SSCC TD Simulated Results

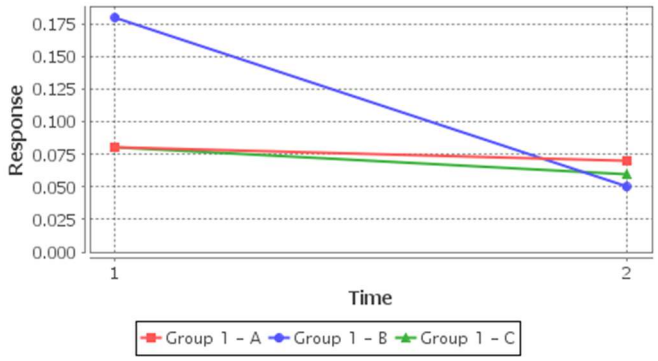
To contrast the capabilities of Q-Learning with the SSCC model we attempted to evaluate an agent which only receives a reward for asking two questions (A and B) and then guessing the animal (C).

We used the following configuration:

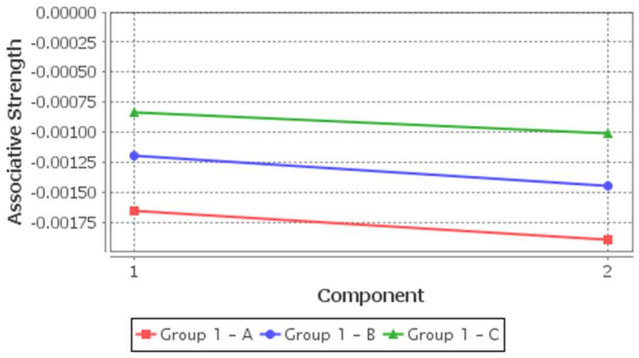
- 100A-/100B-/100C-/100AA-/100AB-/100AC-/100BA-/100BB-/100BC-/100CA-/100CB-/100CC-/AAA-/BAA-/CAA-/AAB-/BAB-/CAB-/AAC-/BAC+/CAC-/ABA-/BBA-/CBA-/ABB-/BBB-/CBB-/ABC+/BBC-/CBC-/ACA-/BCA-/CCA-/ACB-/BCB-/CCB-/ACC-/BCC-/CCC-/
    - i.e. we attempted associative learning for either A, B then C, or B, A then C
- 2 second stimulus duration

Our initial results were as follows however there was not time to complete further evaluation.

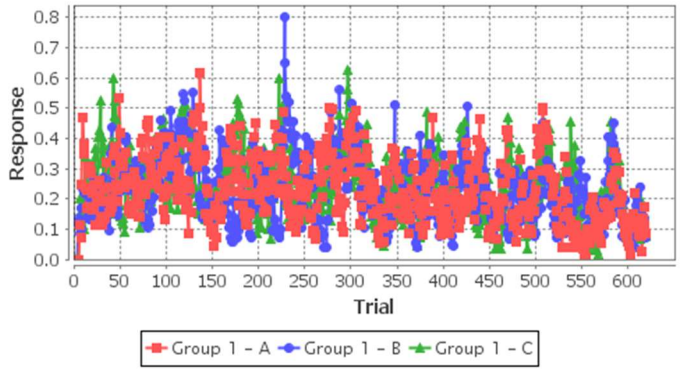
Responses per CS Component



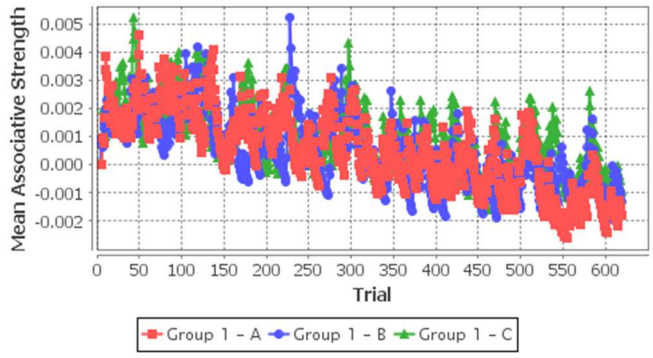
Value per CS Component



Responses per Trial



Value per Trial



Appendix B

SUMMARY OF BASELINE GRIDSEARCH (AVERAGE OF 8 SEARCHES)

STABILISED REWARDS

Row Labels	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
Ep- Midpoint 0.01	**-0.0588**	**-0.1113**	**-0.0712**	**0.0775**	**-0.0213**
Alpha 0.05	-0.3150	-0.3263	-0.3037	-0.3238	-0.3300
0.20	-0.2788	-0.2575	-0.2938	-0.2938	-0.2575
0.50	-0.2875	-0.2613	-0.2500	-0.1313	-0.1713
0.80	-0.0938	-0.2588	-0.2163	-0.0250	-0.0213
0.95	-0.0588	-0.1113	-0.0712	0.0775	-0.0913
0.05	**-0.0675**	**-0.0663**	**-0.0813**	**0.1662**	**-0.0150**
0.05	-0.4000	-0.2975	-0.3275	-0.2650	-0.2925
0.20	-0.2750	-0.2550	-0.2663	-0.1050	-0.2600
0.50	-0.2613	-0.2988	-0.2688	-0.1588	-0.0150
0.80	-0.1538	-0.2375	-0.0813	-0.0550	-0.1325
0.95	-0.0675	-0.0663	-0.1363	0.1662	-0.1400
0.10	**-0.1588**	**-0.2775**	**-0.0988**	**0.0863**	**0.0987**
0.05	-1.5300	-0.8025	-0.2688	-0.3213	-0.3113
0.20	-0.2138	-0.2850	-0.2363	-0.2713	-0.2013
0.50	-0.2650	-0.2825	-0.2625	-0.2263	-0.0363
0.80	-0.3100	-0.7350	-0.0988	0.0863	-0.1212
0.95	-0.1588	-0.2775	-0.1800	-0.0850	0.0987
0.20	**-0.2150**	**-0.2563**	**-0.2500**	**-0.0725**	**-0.0300**
0.05	-2.0637	-1.3188	-0.5700	-0.2288	-0.1988
0.20	-0.2150	-0.3450	-0.2500	-0.2213	-0.1675
0.50	-1.8100	-0.2563	-0.5725	-0.1488	-0.0900
0.80	-1.7600	-1.9125	-1.1512	-0.0850	-0.1588
0.95	-1.4125	-1.2475	-1.0588	-0.0725	-0.0300
0.30	**-0.8650**	**-0.6013**	**-0.7850**	**-0.4388**	**0.0313**
0.05	-5.9388	-3.1350	-2.1613	-0.5575	-0.4000
0.20	-0.8650	-0.6013	-0.7850	-0.4388	-0.4425
0.50	-2.4475	-2.0238	-2.8175	-1.0000	0.0313
0.80	-3.6188	-4.6250	-3.4338	-1.2488	-1.3825
0.95	-4.9750	-4.2575	-3.4238	-1.5700	-1.3163

FINAL EPISODES

Row Labels	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**18,078**	**16,761**	**15,428**	**11,122**	**9,851**
0.05	18,078	15,845	14,111	10,700	9,851
0.20	10,334	10,669	9,072	7,919	7,611
0.50	11,895	11,993	9,832	8,828	7,778
0.80	16,571	16,761	15,428	9,143	8,676
0.95	16,482	15,727	15,079	11,122	9,254
0.05	**19,360**	**18,359**	**17,464**	**12,189**	**11,684**
0.05	19,360	18,022	15,749	12,080	11,684
0.20	12,914	12,994	11,422	9,930	9,679
0.50	13,955	14,128	12,578	10,792	10,338
0.80	17,485	18,359	16,393	12,189	11,044
0.95	17,869	17,849	17,464	11,957	10,783
0.10	**20,000**	**19,398**	**18,658**	**15,338**	**13,612**
0.05	20,000	19,355	16,847	13,868	13,373
0.20	14,637	14,438	12,867	11,803	12,580
0.50	16,389	15,795	13,989	12,582	12,639
0.80	19,427	19,356	17,832	14,105	13,452
0.95	19,412	19,398	18,658	15,338	13,612
0.20	**20,000**	**20,000**	**19,990**	**18,505**	**18,332**
0.05	20,000	20,000	19,652	16,665	16,876
0.20	17,861	18,548	16,928	15,801	16,279
0.50	19,336	19,198	18,200	16,669	16,800
0.80	20,000	20,000	19,915	18,505	18,332
0.95	20,000	20,000	19,990	18,490	17,469
0.30	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	19,928	19,836
0.20	20,000	19,933	19,823	19,773	19,832
0.50	20,000	20,000	20,000	19,755	19,647
0.80	20,000	20,000	20,000	19,997	20,000
0.95	20,000	20,000	20,000	20,000	20,000

ACCURACY

Row Labels	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**1.0000**	**0.9868**	**0.9934**	**1.0000**	**1.0000**
0.05	0.9934	0.9539	0.9803	0.9934	0.9803
0.20	0.9737	0.9671	0.9934	1.0000	1.0000
0.50	1.0000	0.9803	0.9934	0.9803	1.0000
0.80	0.9868	0.9737	0.9671	1.0000	1.0000
0.95	0.9737	0.9868	0.9605	1.0000	0.9934
0.05	**0.9934**	**0.9868**	**1.0000**	**1.0000**	**1.0000**
0.05	0.9803	0.9803	0.9868	0.9934	0.9934
0.20	0.9934	0.9803	1.0000	1.0000	1.0000
0.50	0.9868	0.9868	0.9934	1.0000	1.0000
0.80	0.9342	0.9868	1.0000	1.0000	1.0000
0.95	0.9671	0.9737	0.9211	0.9671	1.0000
0.10	**0.9868**	**0.9934**	**1.0000**	**1.0000**	**1.0000**
0.05	0.9145	0.8750	0.9868	0.9868	1.0000
0.20	0.9474	0.9671	1.0000	1.0000	1.0000
0.50	0.9868	0.9934	0.9868	1.0000	1.0000
0.80	0.9737	0.9408	0.9934	1.0000	0.9671
0.95	0.9737	0.9737	0.9605	1.0000	0.9934
0.20	**0.9605**	**0.9868**	**0.9934**	**1.0000**	**1.0000**
0.05	0.9079	0.9211	0.9671	1.0000	1.0000
0.20	0.9605	0.9868	0.9803	1.0000	1.0000
0.50	0.9342	0.9803	0.9934	1.0000	1.0000
0.80	0.8947	0.8882	0.9474	1.0000	0.9934
0.95	0.8882	0.9671	0.9868	0.9934	1.0000
0.30	**0.9868**	**0.9605**	**0.9671**	**0.9934**	**1.0000**
0.05	0.6250	0.8092	0.8553	0.9934	1.0000
0.20	0.9868	0.9605	0.9671	0.9342	1.0000
0.50	0.8553	0.7697	0.7105	0.9539	1.0000
0.80	0.8092	0.7434	0.7895	0.9145	1.0000
0.95	0.6184	0.7434	0.7368	0.9276	0.9868

AVERAGE QUESTIONS

Row Labels	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**4.9803**	**5.1250**	**5.1711**	**5.0395**	**5.0592**
0.05	5.1513	5.2039	5.1711	5.3224	5.2434
0.20	5.2697	5.2303	5.2566	5.2632	5.3224
0.50	5.2171	5.2566	5.1908	5.0987	5.1579
0.80	5.0855	5.1645	5.1908	5.0987	5.0592
0.95	4.9803	5.1250	5.1711	5.0395	5.1382
0.05	**5.0461**	**4.9868**	**5.1184**	**4.8553**	**5.0066**
0.05	5.1908	5.1513	5.2500	5.3158	5.3026
0.20	5.2303	5.2171	5.3289	5.2171	5.2961
0.50	5.2303	5.2105	5.2763	5.1842	5.0066
0.80	5.0461	5.2237	5.1184	5.1250	5.2171
0.95	5.1184	4.9868	5.2039	4.8553	5.1645
0.10	**5.0132**	**4.8947**	**5.1974**	**5.0000**	**4.9408**
0.05	5.1579	5.0658	5.2961	5.2303	5.3092
0.20	5.2434	5.2303	5.3158	5.2895	5.2895
0.50	5.3289	5.2105	5.2895	5.2632	5.0724
0.80	5.2368	5.1711	5.1974	5.0000	5.0132
0.95	5.0132	4.8947	5.2566	5.0789	4.9408
0.20	**5.0724**	**5.0066**	**5.0526**	**5.1118**	**5.1053**
0.05	5.2368	5.1184	5.3026	5.2895	5.2895
0.20	5.2697	5.2961	5.2632	5.2829	5.2303
0.50	5.0921	5.2434	5.2303	5.2303	5.1645
0.80	5.0724	5.0066	5.0921	5.1118	5.1382
0.95	5.1053	5.1316	5.0526	5.1316	5.1053
0.30	**4.6974**	**4.7763**	**4.8158**	**4.8092**	**4.9934**
0.05	4.6974	5.0329	5.0658	5.3026	5.2829
0.20	5.2961	5.2895	5.2763	5.0395	5.3158
0.50	4.9868	4.7763	4.8158	5.1711	4.9934
0.80	5.0789	4.9671	5.0066	4.8092	5.1645
0.95	4.9145	4.9737	4.9211	5.0987	5.0592

Appendix C

SUMMARY OF BINARY GRIDSEARCH (AVERAGE OF 8 SEARCHES)

STABILISED REWARDS

Ep- Midpoint \ Alpha	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**3.2667**	**3.8667**	**3.8000**	**3.3667**	**3.7667**
0.05	0.6667	1.0000	1.2333	1.4333	1.3333
0.2	1.1667	2.3667	2.9333	2.7667	3.7667
0.5	2.6667	3.5333	3.0333	3.3667	3.2000
0.8	2.8000	3.8667	3.8000	3.2667	2.9333
0.95	3.2667	2.5000	3.6000	2.6000	3.1667
0.05	**2.8000**	**3.1667**	**4.0000**	**3.2333**	**3.9333**
0.05	0.7667	1.1667	1.1667	1.4000	1.4333
0.2	1.4000	1.8333	2.6667	2.8333	3.9333
0.5	2.7667	2.7000	2.5000	3.2000	2.6667
0.8	1.9667	3.1667	2.2667	3.2333	3.8667
0.95	2.8000	3.1333	4.0000	3.0667	2.5667
0.1	**3.1333**	**3.9000**	**3.9667**	**3.8667**	**3.8333**
0.05	0.4333	0.6333	1.3333	1.5667	1.6667
0.2	1.5000	2.1667	2.6667	3.4667	3.1333
0.5	2.0000	3.1667	3.9667	3.8667	3.8333
0.8	2.8000	3.9000	3.1333	3.8667	3.8000
0.95	3.1333	3.3000	3.4333	3.0333	2.6667
0.2	**3.4667**	**3.7667**	**3.6667**	**3.9667**	**4.1333**
0.05	0.6667	0.8000	0.9667	1.7000	1.4667
0.2	1.3000	1.3667	2.7667	3.9333	4.1333
0.5	2.3000	3.0000	3.6667	3.9333	3.8333
0.8	3.1667	3.5667	3.3667	3.9667	3.1667
0.95	3.4667	3.7667	3.4333	3.9667	3.5333
0.3	**3.0000**	**3.1333**	**3.3667**	**3.7000**	**3.5667**
0.05	0.4667	0.8000	0.9000	1.5333	2.5000
0.2	1.3667	1.8667	1.7333	2.9667	2.9333
0.5	1.5667	1.9333	3.3667	3.7000	2.9333
0.8	3.0000	2.9667	2.9333	3.5667	2.4333
0.95	2.6000	3.1333	3.1333	3.6000	3.5667

FINAL EPISODES

Ep- Midpoint \ Alpha	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	20,000	20,000
0.2	20,000	20,000	20,000	20,000	20,000
0.5	20,000	20,000	20,000	20,000	20,000
0.8	20,000	20,000	20,000	20,000	20,000
0.95	20,000	20,000	20,000	20,000	20,000
0.05	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	20,000	20,000
0.2	20,000	20,000	20,000	20,000	20,000
0.5	20,000	20,000	20,000	20,000	20,000
0.8	20,000	20,000	20,000	20,000	20,000
0.95	20,000	20,000	20,000	20,000	20,000
0.1	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	20,000	20,000
0.2	20,000	20,000	20,000	20,000	20,000
0.5	20,000	20,000	20,000	20,000	20,000
0.8	20,000	20,000	20,000	20,000	20,000
0.95	20,000	20,000	20,000	20,000	20,000
0.2	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	20,000	20,000
0.2	20,000	20,000	20,000	20,000	20,000
0.5	20,000	20,000	20,000	20,000	20,000
0.8	20,000	20,000	20,000	20,000	20,000
0.95	20,000	20,000	20,000	20,000	20,000
0.3	**20,000**	**20,000**	**20,000**	**20,000**	**20,000**
0.05	20,000	20,000	20,000	20,000	20,000
0.2	20,000	20,000	20,000	20,000	20,000
0.5	20,000	20,000	20,000	20,000	20,000
0.8	20,000	20,000	20,000	20,000	20,000
0.95	20,000	20,000	20,000	20,000	20,000

ACCURACY

Ep- Midpoint \ Alpha	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**0.3158**	**0.3684**	**0.4035**	**0.3684**	**0.3684**
0.05	0.0702	0.1053	0.1053	0.1579	0.1579
0.2	0.1579	0.2281	0.2982	0.2982	0.3684
0.5	0.2456	0.3509	0.2982	0.3684	0.3158
0.8	0.2807	0.3684	0.4035	0.3158	0.2982
0.95	0.3158	0.2807	0.3333	0.2281	0.3333
0.05	**0.3158**	**0.3509**	**0.3860**	**0.3684**	**0.3860**
0.05	0.0877	0.1053	0.1053	0.1228	0.1579
0.2	0.1404	0.1754	0.2632	0.2807	0.3684
0.5	0.3158	0.3509	0.2982	0.3509	0.2807
0.8	0.2632	0.2982	0.2456	0.3333	0.3860
0.95	0.2632	0.3158	0.3860	0.3684	0.3158
0.1	**0.3509**	**0.3333**	**0.3684**	**0.3684**	**0.4211**
0.05	0.0526	0.0877	0.1053	0.1579	0.1754
0.2	0.1404	0.2281	0.2632	0.3333	0.3158
0.5	0.2105	0.3158	0.3509	0.3684	0.3509
0.8	0.2456	0.3333	0.3684	0.3509	0.4211
0.95	0.3509	0.3158	0.3333	0.3333	0.2982
0.2	**0.3684**	**0.4386**	**0.4035**	**0.4386**	**0.4386**
0.05	0.0526	0.0702	0.1053	0.1579	0.1404
0.2	0.1579	0.1579	0.2632	0.3684	0.4211
0.5	0.2456	0.2807	0.4035	0.3333	0.4386
0.8	0.3684	0.3860	0.3333	0.4386	0.2281
0.95	0.3333	0.4386	0.4035	0.4035	0.3509
0.3	**0.3684**	**0.3509**	**0.3333**	**0.4211**	**0.3860**
0.05	0.0526	0.0702	0.1053	0.1579	0.2456
0.2	0.1404	0.1579	0.2281	0.2982	0.2982
0.5	0.1404	0.1930	0.3333	0.4211	0.3860
0.8	0.3684	0.2807	0.2632	0.2807	0.3333
0.95	0.2807	0.3509	0.3333	0.3684	0.3509

AVERAGE QUESTIONS

Ep- Midpoint \ Alpha	Gamma (columns) 0.05	0.2	0.5	0.8	0.95
0.01	**0.3333**	**1.0000**	**1.0000**	**1.5965**	**1.7018**
0.05	0.3333	1.0000	1.0000	1.5965	1.7018
0.2	1.5263	2.1228	2.7544	2.6842	2.9298
0.5	2.2105	3.1404	2.4561	2.7719	2.8070
0.8	2.5263	2.9474	3.0526	2.6140	2.7193
0.95	2.6140	2.5789	2.7544	1.7544	2.7018
0.05	**0.6667**	**1.0000**	**1.0000**	**1.2105**	**1.5439**
0.05	0.6667	1.0000	1.0000	1.2105	1.5439
0.2	1.4211	1.7193	2.3509	2.3158	3.1930
0.5	2.5789	2.8246	2.6316	3.1754	2.3684
0.8	2.5088	2.7895	2.2281	2.8070	3.1053
0.95	1.8070	2.6842	2.8596	2.8070	2.5965
0.1	**0.0000**	**0.6667**	**0.8772**	**1.6491**	**2.0000**
0.05	0.0000	0.6667	0.8772	1.6491	2.0000
0.2	1.3333	2.1228	2.3333	2.8421	2.7544
0.5	2.0702	2.6491	3.0702	3.1228	2.9298
0.8	2.0877	2.5614	3.1579	2.9649	3.3509
0.95	2.7368	2.8421	2.8596	2.7193	2.7719
0.2	**0.0000**	**0.3333**	**1.0000**	**1.7368**	**1.4561**
0.05	0.0000	0.3333	1.0000	1.7368	1.4561
0.2	1.6667	1.6316	2.2456	2.7895	3.5263
0.5	2.2807	2.4035	3.1930	2.4912	3.1930
0.8	2.9825	3.0000	2.6842	3.1404	1.7719
0.95	3.0175	3.4737	3.1754	3.1930	2.7368
0.3	**0.0000**	**0.3333**	**0.8772**	**1.7544**	**2.1053**
0.05	0.0000	0.3333	0.8772	1.7544	2.4211
0.2	1.4211	1.5965	2.0877	2.4035	2.1053
0.5	1.2632	1.8772	2.9298	3.2982	3.0702
0.8	2.8070	2.7193	2.7544	2.2456	2.9649
0.95	2.4737	2.8070	2.9298	3.1579	3.0351

References

1 Kaelbling, L.P., Littman, M.L. & Moore, A.W. 1996, "Reinforcement Learning: A Survey", Journal of Artificial Intelligence Research, vol. 4, pp. 237-285.

2 Zhao, T. & Eskenazi, M. 2016, "Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning".

3 Chen, Y., Chen, B., Duan, X., Lou, J., Wang, Y., Zhu, W. & Cao, Y. 2018, "Learning-to-Ask: Knowledge Acquisition via 20 Questions".

4 Hu, H., Wu, X., Luo, B., Tao, C., Xu, C., Wu, W. & Chen, Z. 2018, "Playing 20 Question Game with Policy-Based Reinforcement Learning".

5 Sutton, R.S., Barto, A.G. & Project MUSE Open Access Books 1998, Reinforcement learning: an introduction, MIT Press, Cambridge, Mass.

6 Eduardo Alonso, Esther Mondragón Class Notes.

7 Liu, Z. & Abbaszadeh, S. 2019, "Double Q-Learning for Radiation Source Detection", Sensors (Basel, Switzerland), vol. 19, no. 4, pp. 960.

8 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. 2013, "Playing Atari with Deep Reinforcement Learning".

9 Ranzato, M., Chopra, S., Auli, M. & Zaremba, W. 2015, "Sequence Level Training with Recurrent Neural Networks".

10 Alonso, E. and Mondragón, E., 2006. Associative Learning for Reinforcement Learning: where animal learning and machine learning meet. In Proceedings of the fifth symposium on adaptive agents and multi-agent systems (pp. 87-99).

11 Veksler, V.D., Myers, C.W. & Gluck, K.A. 2014, "SAwSu: An Integrated Model of Associative and Reinforcement Learning", Cognitive Science, vol. 38, no. 3, pp. 580-598.

12 Kokkola, N.H., Mondragón, E. & Alonso, E. 2019, "A double error dynamic asymptote model of associative learning", Psychological review.

13 Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. Machine learning, 3(1), pp.9-44.

14 Nasser, H.M., Calu, D.J., Schoenbaum, G. & Sharpe, M.J. 2017, "The Dopamine Prediction Error: Contributions to Associative Models of Reward Learning", Frontiers in psychology, vol. 8, pp. 244.

15 Mondragón, E., Alonso, E. & Kokkola, N. 2017, "Associative Learning Should Go Deep", Trends in Cognitive Sciences, vol. 21, no. 11, pp. 822-825.